Experiments in Fluids 30 (2001) 399–409 © Springer-Verlag 2001

# Neural network reconstruction of fluid flows from tracer-particle displacements

**G. Labonté**



**Abstract** We demonstrate some of the advantages of using artificial neural networks for the post-processing of particle-tracking velocimetry (PTV) data. This study is concerned with the data obtained after particle images have been matched and the obvious outliers have been removed. We show that it is easy to produce simple back-propagation neural networks that can filter the remaining random noise and interpolate between the measurements. They do so by performing a particular form of non-linear global regression that allows them to reconstruct the fluid flow for the entire field covered by the photographs. This is obtained by training these neural networks to learn the fluid dynamics function **f** that maps the position **x** of a fluid particle at time $t$ to its position **X** at time $t + \Delta t$. They can do so with a high degree of precision when provided with pairs of matching particle positions (**x**, **X**) from only about 2 to 4 pairs of PTV photographs as exemplars. We show that whether they are trained on exact or on noisy data, they learn to interpolate with such a precision that their output is within one pixel of the theoretical output. We demonstrate their accuracy by using them to draw whole streamlines or flow profiles, by iteration from a single starting point.

## 1 Introduction

Particle-tracking velocimetry (PTV) is a technique used in particle-image velocimetry (PIV) for determining the velocity field of a fluid in motion [see Adrian (1991)]. It consists in measuring the displacements undergone by small particles suspended in this fluid during a small time interval $\Delta t$. In its simpler form, this technique is mainly used for two-dimensional fluid flows. Two photographs of the same region of the fluid are taken, from a direction at 90° to its plane of motion, one at time $t$ and the other one at time $t + \Delta t$. These are then usually digitized so that a computer can automatically process them. It is necessary to find the position of the geometric center of the particle images, in some reference frame. There are many efficient procedures for doing so, e.g., the neural network method described by Carosone et al. (1995). Once this is done, one must solve the correspondence problem, which consists in determining which points, in the two photographs, represent the same particles. Labonté (1999) mentions some of the best methods used for this and describes a self-organizing map neural network that can very efficiently solve this problem. The present work deals with the processing of the data obtained once image matching is completed.

### 1.1 The inaccuracies in the data

As discussed, e.g., in Adrian (1991), Westerweel (1994) and Luff et al. (1999), the errors in the measurements of PTV and, more generally of PIV, are essentially of two kinds. There are errors that result from false matches between particle pairs in PTV and, with correlation methods in PIV, from the selections of false correlation peaks. These errors yield spurious vectors in the velocity field. The tip of such vectors is at a random, uniformly distributed, position in the interrogation window (of PTV or PIV). Many of them are thus much larger than their neighbors or have very different directions from them so that they can easily be recognized as outliers. Figure 1 illustrates this point; it shows displacement vectors as would be obtained in PTV, after matching two photographs of 200 particles each, each one having 20 particles that are unmatchable due to out of plane motion. The flow is that in a vortex with complex potential $V(z) = i/z$ that is singular at the origin. The number of spurious vectors present evidently depends on the efficiency of the matching algorithm. For the singular flow shown in Fig. 1, which is a difficult matching problem, there were 24 outliers out of 180 vectors, i.e., about 13% of outliers. Westerweel (1994) mentions the figure of 5% as typical for PIV data, and Sun et al. (1996) gives 15% for a difficult problem with high speed flow in a combustion chamber.

The other kind of errors in PIV result from the preprocessing of the pictures, when the background is removed and the intensity of the particle images is enhanced, from the finiteness of the grain of the film or of the pixel size in light detector arrays, and from the noise in the electronic device that digitizes the data. With correlation methods, there are also inaccuracies due to the deformation of particle-image patterns. A small number of

*Received: 23 November 1998/Accepted: 14 July 2000*

G. Labonté
Department of Mathematics and Computer Science
Royal Military College of Canada, Kingston
Ontario K7K 7B4, Canada
e-mail: labonte-g@rmc.ca
Fax: +1-613-3845792

This research was financed in part by a grant from the Academic Research Program of the Department of National Defense of Canada.

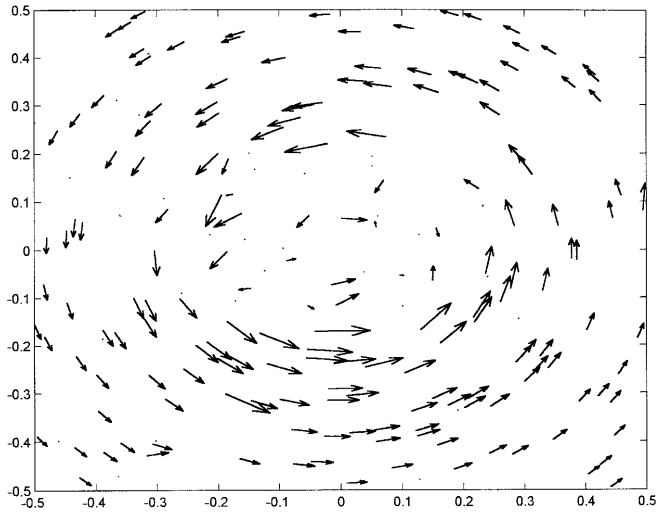

**Fig. 1.** Displacement vectors as obtained in PTV, after matching particle images in two photographs of 200 particle images. The flow is that of a singular vortex centered in the middle of the picture. Many spurious vectors are easily recognized

errors also correspond to the spurious vectors that were left in, after the attempt at their removal, because they were very close to the actual vectors. All of these errors are much smaller than those corresponding to outliers and constitute essentially random noise. In PTV, they lead to measured positions of the centroids of the particles $\{X_i'$: $i = 1,\ldots,N\}$, being off by some random error vector $\varepsilon_i$ so that, even in a correctly matched image pair, the measured displacement $D_i'$ is actually $\mathbf{D}_i + \varepsilon_i$, where $\mathbf{D}_i$ is the true displacement. One can assume as in Westerweel (1994) that the components of $\varepsilon_i$ are statistically orthogonal and are distributed normally about the value 0 with standard deviation $\sigma_\varepsilon$. This author mentions that, for most PIV systems, $\sigma_\varepsilon$ is estimated at less than 1% of the full displacement range, which is defined as = (length of the maximum displacement) – (length of the minimum one). Huang et al. (1993) mentions that locating the center of intensity of particles in PIV is usually done with an error under 0.25 pixels. Luff et al. (1999) test their data processing methods with artificial data having random, uniformly distributed, 1%, 3.3% and 4.5% relative absolute error in the velocity vectors. For particle-image displacements of 10–20 pixels, the largest errors are then about 0.5–0.9 pixel. These figures are somewhat higher than those of other authors, as quoted in Guézennec and Kiritsis (1990). Guézennec et al. (1994) mention that in their tests a sub-pixel accuracy of the average of 0.3 pixel is normal.

## 1.2
## Spurious vector removal

A large proportion of the spurious vectors can easily be eliminated manually, as mentioned in Westerweel (1994) and Sun et al. (1996). Upon so doing with Fig. 1, we removed successfully 20 out of the 24 spurious vectors. The four that we did not remove happened to be, by accident, close enough to the correct displacement vectors that they could be mistaken for these. This similarity to actual displacement vectors implies that the continuity of the displacement field will not be much altered by their being left in. Efficient algorithms, based on the continuity of the displacement vector field, exist to remove the outliers; see for example Landreth and Adrian (1990), Willert and Gharib (1991), Guézennec et al. (1994), Westerweel (1994), Hartman et al. (1996), and Song et al. (1999). As in the case of manual outlier removal, these methods are efficient enough that the only missed vectors are close to the actual displacement vectors and the displacement field is not appreciably distorted by their being left in.

## 1.3
## Random noise removal

Once outliers have been removed, there is often need for further processing. This may be done to improve the accuracy of physical variables that are to be calculated from the velocity field, such as the vorticity (see Lourenco et al. 1995; Sun et al. 1996; Luff et al. 1999). It can also be done for the important purpose of flow field visualization, in which case, post-processing aims at improving the appearance for human observers of the graphical representations of the velocity field. Algorithms are then needed for the smoothing of the data and its interpolation.

In most published work in PIV, smoothing of the data is done simply by convolving the velocity field with a Gaussian function (see, e.g., Luff et al. 1999; Sun et al. 1996; Hartman et al. 1996). This technique usually succeeds in filtering out some of the random noise. Luff et al. (1999) report results to that effect for data obtained by auto-correlation in PIV. In Hartman et al. (1996), the Gaussian filter is combined with an extended vector median filter, as described in Astola et al. (1990). The latter filter is used mainly to remove the outliers but it does at the same time cancel some of the random noise.

One should however be cautious with filters because there will actually be a loss, instead of a gain, of precision when they average or convolve data over too large a domain. Small-scale information is then destroyed by their blurring effect (see Luff et al. 1999; Astola et al. 1990).

## 1.4
## Interpolation

Interpolation is used in post-processing mainly for filling up the holes left by outlier removal or where the matching algorithm found no matches. It is then usually done only locally for small patches of the displacement field. The interpolation techniques normally used are spline fitting and linear regression to least-square fit with some chosen functions. Typically, Sun et al. (1996) use quadratic functions to interpolate the missing velocity vectors and Luff et al. (1999) do local regression using a second-degree polynomial. For data visualization, they use spline fitting, as do most commercial software.

Attempts are rarely made at fitting the whole or a large region of the displacement field. Luff et al. (1999) show the result they obtained when doing a full field regression, with a fourth degree polynomial, for the vorticity field of an Oseen vortex. The poor quality of the fit obtained is evident; the maximum relative error is 226% and the surface average relative error is 55%.

## 2
## Smoothing and interpolation by neural networks

Neural networks are massively parallel data processing systems made up of a large number of very simple processing units. These individual processors are, most of the time, modeled after the McCulloch and Pitts' (1943) artificial neuron, also called the perceptron neuron. It is schematically represented as a cell (processor) that receives a certain number of input leads and has only one output lead. Let the $n$ variable inputs be represented by $x_1$, $x_2$, …, $x_n$. Often one wants to add an input bias term; this is done by adding a constant input $x_0$ set to 1. Each input lead has a variable transmission coefficient, analogous to a resistance, called a weight, denoted here by $w_0$, $w_1$, …, $w_n$. When receiving an input, the processor sums up its total signal as $I = \sum_{i=0}^{n} w_i x_i$. Upon using column vectors $\mathbf{x}$ and $\mathbf{w}$ to denote the inputs and the weights, I can be more simply written as $\mathbf{w}^T\mathbf{x}$, where $\mathbf{w}^T$ is the transposed of $\mathbf{w}$. The processor's transfer function $f$ is usually a sigmoid function, that is, a 'softened' Heaviside step function, such as a logistic, an hyperbolic tangent or an inverse tangent function. It is often also simply a linear function. The neuron output is $O = f(\mathrm{I})$. The neurons in a neural network usually have different weights $\mathbf{w}$, while many of them would have the same transfer function. They are interconnected in different patterns, the output of one becoming a part of the input of many others. Artificial neural networks learn and adapt through the modification of their weights, like the natural ones were shown to do by Hebb (1949).

### 2.1
### Neural network filters

Filters that correspond to the convolution of a constant mask with data provided at points in a regular array, as with the displacement vectors obtained by correlation in PIV, are easily implemented with neural networks. Such a neural network would have a single sheet of neurons arranged regularly, one neuron corresponding to each array point. The biological analogy of this system is a retina that provides the input to a neural network consisting of a layer of neurons. Figure 2 shows the connection pattern of this system. Each neuron has the same constant weights, which we denote here $w_{m,n}$, $m$ and $n$ taking all the integer values between $-k$ and $+k$. The output of the neuron at position $(i, j)$ is

$$O_{ij} = \sum_{m,n=-k}^{k} w_{mn} x_{i+m,j+n}$$

and it represents the filtered value at that position of the data array.

The advantage of using a neural network to calculate this convolution resides in the short processing time it requires. If the number of array points of the data is $P$, and $T_0$ is the time required for one multiplication and one addition, the neural network will perform the filtering of the data essentially in the constant time $T_0$. The same filtering operation done sequentially would require the computing time $P2kT_0$. There could be a disadvantage in that the neural network requires $P$ processing units, the same number as there are data points, were it not for the fact that these processors are all identical, extremely simple and supposedly very cheap to manufacture. For a photograph divided in $50 \times 50$ patches, 2,500 neurons are required and, for a $3 \times 3$ filter mask, the computation time is of the order of 45,000 times shorter than that of the equivalent sequential computation.

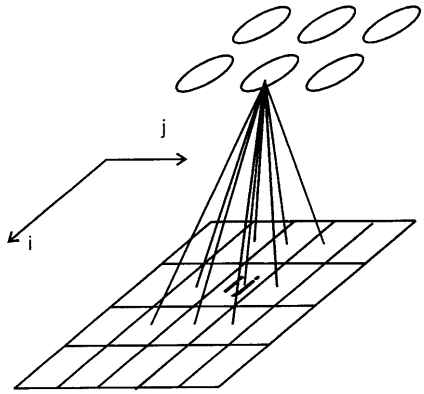


**Fig. 2.** A neural network that filters data. The bottom square grid represents the array of data, as provided by a "retina". The array of circles above it represents the network of perceptron neurons that form a sheet and implement the filter. Neuron number $(i, j)$, which is above array point $(i, j)$, is connected to the array points in the neighborhood of this point. The weight on the connection to point $(I + m, j + n)$ is $w_{mn}$

### 2.2
### Neural networks for linear regression and spline fitting

The regressions normally done in PIV data post-processing are local linear regressions. One is given a set of data points $(\mathbf{x}_1, \mathbf{y}_1)$, $(\mathbf{x}_2, \mathbf{y}_2)$, …, $(\mathbf{x}_n, \mathbf{y}_n)$, where $\mathbf{y}_1$, …, $\mathbf{y}_n$ are measurement values at the points $\mathbf{x}_1$, …, $\mathbf{x}_n$ which span a domain $D$. One then selects a set of functions $\{f_1, f_2, \ldots, f_k\}$ and looks for parameters $c_1$, $c_2$, …, $c_k$ such that in the domain $D$ the function $F$:

$$F(\mathbf{x}) = \sum_{m=1}^{k} c_m f_m(\mathbf{x}) \qquad (1)$$

defined $\forall\, \mathbf{x} \in D$, provides a least square fit to the data. The parameters $c_1$, …, $c_k$ must then be such that the total quadratic error

$$E_q = \sum_{i=1}^{n} \|y_i - F(x_i)\|^2 \qquad (2)$$

is minimum. $N$th degree spline fitting is obtained when the functions $f_m$ are the monomials of degree up to $N$ in the components of $\mathbf{x}$ and when $n$, the number of data points, equals $k$, the number of these monomials. More generally, the functions $f_m$ can be any functions one finds appropriate for a given situation, such as members of a set of orthogonal functions (as in partial Fourier series), Gaussian functions centered at different points, functions known as Hardy (1971) multiquadrics, etc.

Because $E_q$ is a quadratic function of the $c_i$, it has a single minimum. The value of $\mathbf{c}$ at this point can be found exactly by solving the set of linear equations $\vec{\nabla} E_q = \vec{0}$.

There is a very efficient two-layer neural network that solves this problem with the help of the LMS (least mean squares) algorithm of Widrow and Hoff (1960). Figure 3 shows its structure. It has $k$ neurons, in its first layer, which are somewhat different from the perceptron neuron. Each one has one of the $f_m$ as its transfer function and for the input $\mathbf{x}$, it simply returns $f_m(\mathbf{x})$. The second layer has a single perceptron neuron, with weight vector $\mathbf{w} = (0, c_1, \ldots, c_k)^T$. Its transfer function is the identity, so that its output is $O = \sum_{m=1}^{k} c_m f_m(\mathbf{x})$. Applications where this artificial neural network is used for linear regression are innumerable; Widrow and Stearns (1985) discusses a certain number of them.



Local regression, on $P$ patches of PIV data, can be performed in parallel by a neural network made up of $P$ such sub-networks as described above. The total number of neurons that would be required is $P(k + 1)$. Therefore, 27,500 neurons would be required for a data set of $50 \times 50$ patches to be fitted with cubic splines, for which $k = 10$. The speed-up can be roughly evaluated to be about 25,000 as follows. The data in all patches would be fitted simultaneously by the 2,500 sub-networks. Furthermore, in each sub-network, the evaluation of the $mf_i(\mathbf{x})$, for $i = 1, \ldots, m$, is done in parallel and so is their subsequent multiplication by $c_i$ and the final summation. Thus, each sub-network by itself provides a speed-up of $m = 10$ over a processing unit that would compute sequentially.

## 2.3
## Neural networks for non-linear regression

Artificial neural networks are also able to perform a type of non-linear regression that, after many years of usage in impressive applications, has been clearly demonstrated to be much more powerful than linear regression. As the title of Werbos' (1974) PhD thesis indicates, they go "Beyond Regression". An excellent discussion of their abilities in pattern recognition and of the advantages they provide can be found in Bishop (1997). As a simple illustration of what is meant here by generalized regression, consider the problem of finding a function $F$ of one variable that would fit a data set $(x_1, y_1), \ldots, (x_n, y_n)$. A 3-layer neural network, in which the hidden layer's transfer function would be sinus, could solve this problem by finding constants $c_1, \ldots, c_k$ and frequencies $w_1, \ldots, w_k$ such that

$$F(x) = \sum_{m=1}^{k} c_m \sin(w_m x)$$

provides the best least square fit to these data points. The frequencies $w_m$ will in general not be simply harmonics of a fundamental frequency with $w_m = mw_0$. Thus, $F$ ends up being a sort of generalized partial Fourier series. The possibility of also adjusting the frequency spectrum of $F$ provides the advantage that the series will not require a very large number of components to fit the data precisely over regions where it varies quickly as well as over regions where it is fairly constant.

Different functions are commonly used in the role of the "sin" in the above sum. Figure 4 shows the structure of a neural network that realizes the function $F$:

$$F(x) = \sum_{m=1}^{k} c_m f\left(\mathbf{w}_m^T \mathbf{x}\right)$$

with perceptron neurons having a transfer function $f$. Neural networks with such patterns of connections are called feed-forward networks. When they are made up of perceptron neurons, they are more specifically referred to as multi-layer perceptrons. As we mentioned before, for such neurons, sigmoids are used most of the time as transfer functions. Some other functions such as Gaussians, which have a more localized action, are also often used. The neurons are then different from the perceptron neuron. Although they also have a weight vector $\mathbf{w}$, when they receive the input $\mathbf{x}$, they will calculate as total input the distance $||\mathbf{w}_m - \mathbf{x}||$, instead of the scalar product $\mathbf{w}_m^T \mathbf{x}$. These functions are commonly referred to as radial basis functions. They are discussed in most textbooks on neural networks, e.g., Hassoun (1995). A detailed analysis

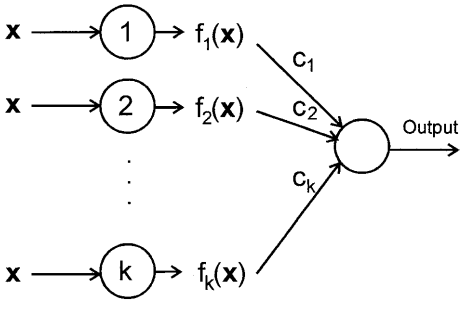


**Fig. 3.** A neural network that performs a linear regression. Neuron $i$ in the input layer has a transfer function $f_i$. Upon receiving the components of the vector $\mathbf{x}$ as input, it outputs $f_i(\mathbf{x})$. The input connections to these neurons simply bring them the components of $\mathbf{x}$ unaltered. We have represented them as a single incoming line for simplicity. The network output is provided by a perceptron neuron with weight vector $\mathbf{c}$ and the identity as its transfer function

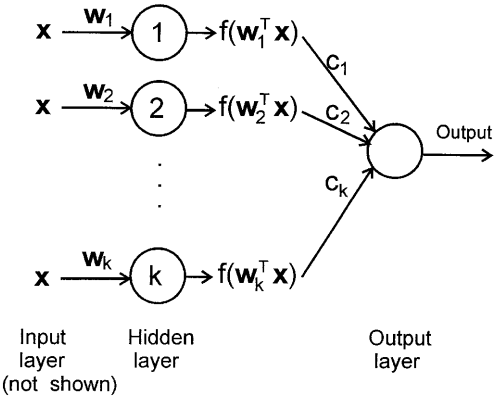


**Fig. 4.** Standard three-layer feed-forward neural network with one output. The input layer distributes the $n$ components of the input vector $\mathbf{x}$ to each of the $N$ neurons in the second layer, called the first hidden layer. The neurons in the hidden layer have $n$ weights, which are represented as the components of the weight-vector $\mathbf{w}$. We have used single input lines, for the neurons in the hidden layer, to represent concisely their $n$ incoming connections. The output of the hidden layer is fed as input to the neuron of the output layer, which has $N$ weights

of them can be found in Poggio and Girosi (1990). With such functions, Eq. (1) could now be understood as spherical harmonic decompositions with varying frequencies.

We now briefly review how a neural network performs this kind of regression. We recall that it has to determine the vector parameters $\mathbf{w}_m$ and the vector of coefficients $\mathbf{c}$, at which the value $E_q(\mathbf{c}, \mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_m)$ of the error function defined in Eq. (2) is minimum. Due to the complexity of the error function, these parameters cannot be solved for exactly. They will be obtained by iteration. The most common and most successful algorithm used for this is error back-propagation [see almost any textbook on neural networks as, e.g., in section 5.3 of Hecht-Nielsen (1989), or for a more detailed account, Almeida (1997)]. This algorithm corresponds to a gradient descent that neural networks realize as follows. For simplicity, let us denote all the weights of the network as a single vector $\mathbf{w}$, so that the quadratic error is $E_q(\mathbf{w})$. The gradient descent is an iterative method such that if the variable at step $t$ is $\mathbf{w}(t)$ then, at step $(t + 1)$, it will be:

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \Delta\mathbf{w}(t)$$

with the change in the weight vector $\Delta\mathbf{w}(t)$ being given by:

$$\Delta\mathbf{w} = -\eta\nabla E_q(\mathbf{w})$$

$\eta$ is a constant called the learning rate and $\nabla$ is the gradient operator.

All points $(\mathbf{x}_i, y_i)$ of the data set are used sequentially to train the neural network as follows. The vector $\mathbf{x}_i$ is presented to it as input; its output $F(\mathbf{x}_i)$ is then observed and the error it made, $|y_i - F(\mathbf{x}_i)|$, is calculated. Because of the layered structure of the neural network, the weight change to be made in the output layer can immediately be calculated from this error. The back-propagation algorithm subsequently provides an efficient method [see section B6.3.3 in Bishop (1997) for a discussion of its efficiency] to perform the calculation of the weight changes for all other layers, moving back from the output layer to the input layer – hence its name of "back-propagation". Using all the data points once is said to constitute a "training epoch" or "cycle". Generally many training cycles are required for the network to learn.

Many variants of back-propagation exist, several of which are based on gradient descent with second order derivatives of the error function $E_q$. A review can be found in Battiti (1992). In the present study, we use the method of Fahlman (1989) called "quick-propagation". It estimates the second derivatives from the differences in the first derivatives from one training epoch to the next.

### 2.4
### Application to PTV

The main purpose of the present study is to test how successful such a neural network can be in using PTV data in which the outliers have been removed to construct a faithful continuous representation of the fluid motion, while canceling the random noise in the data. For a given fluid flow, we will let $\mathbf{f}$ denote the fluid dynamics function that takes the position $\mathbf{x}$ of a fluid particle at time $t$, and transforms it in $\mathbf{X} = \mathbf{f}(\mathbf{x})$, its position at time $t + \Delta t$. The neural network will have to learn to approximate $\mathbf{f}$, to a high degree of accuracy, by generalization from examples of pairs of positions $(\mathbf{x}_i, \mathbf{X}_i)$, corresponding to measurements from the PTV experiment.

## 3
## Existence theorem

A guarantee that a neural network can approximate the fluid dynamics function $\mathbf{f}$ is provided by very general results due, among others, to Cybenko (1989), Hornik et al. (1989) and Funahashi (1989) and discussed in section 2.3 of Hassoun (1995). These results concern three layer networks, as shown in Fig. 4. They have one hidden layer for which the transfer function is some continuous sigmoid function $\sigma$, as illustrated in Fig. 4. The input layer has as many neurons as there are components to the input vectors $\mathbf{x}$. These neurons have fixed weights and their role is simply to distribute the components of $\mathbf{x}$ to each neuron of the next layer. Because of this, they are often not considered explicitly in the discussion of the neural network. The next layer is composed of $N$ neurons, each of which has $n$ weights and a threshold value $s$, represented by its bias weight being set to $-s$. The weights of the $i$th neuron constitute the components of the vector $\mathbf{w}_i$. This neuron then receives the total input $I_i = \mathbf{w}_i^T\mathbf{x}$. Its transfer function is a sigmoid $\sigma$, so that its output is $\sigma(\mathrm{I}_i)$. The output layer has only one neuron, with weight vector $\mathbf{c} \in R^N$, and its transfer function is the identity. Its output is then: $F(\mathbf{x}) = \sum_{i=1}^{N} c_i\sigma(\mathbf{w}_i^T\mathbf{x})$. The theorem in question states that, given any piecewise continuous function $f$, defined on a compact subset $D$ of $R^n$, and any $\varepsilon > 0$, there exists such a neural network for which $|F(\mathbf{x}) - f(\mathbf{x})| < \varepsilon$, $\forall\ \mathbf{x} \in D$. The same theorem also holds true when the output layer has a sigmoid transfer function instead of the identity. It also generalizes to functions $\mathbf{f}$ that have their value in $R^m$, in which case there are $m$ neurons in the output layer.

We note that, even though the theory guarantees that three layers suffice for the neural network, it often happens that networks with four or more layers have to be used, because an adequate three-layer network would require an impracticably large number of neurons in its hidden layer. There are no precise guidelines as to how to select neural network architectures. There exist some partial results, such as that of Schwartz et al. (1990), who have indicated that it should not contain more connections than the number of data samples. However, there is no doubt that the most influential factor is the nature of the function to be approximated, i.e., how fast it varies, etc. A discussion of the main considerations that influence the choice of the neural network parameters can be found in section C1.2 of Almeida (1997). In our study, we wanted also to find out how difficult it actually is to obtain a neural network structure and parameters that can make the network learn the fluid dynamics function $\mathbf{f}$. Thus, we started without using any of these "tricks of the trade". We simply took a relatively inexpensive general-purpose neural network software, selected the option "backprop" as the training algorithm, and used the default settings proposed by the software. For determining the network structure, i.e., its number of layers and neurons per layer, we adopted the strategy of starting with very simple neural networks and

increasing their complexity until they became able to realize the desired function approximation.

### 3.1
### Noise cancellation

Not only can neural networks realize all physically occurring functions, but they also have the further ability to filter out noise from experimental data. This has been demonstrated before by Tamura and Waibel (1988) for speech processing data, and by Troudet and Merrill (1990) for control signals. Bishop (1997) has actually proved that for a very large data set minimizing the quadratic error $E_q$ results in a canceling out of the Gaussian noise. In fact, it actually benefits a neural network to have random noise added to its training data, as it results in an improvement of its robustness (see Noyes 1997).



## 4
## Training procedure

How to train a neural network with experimental data has been well studied; good guidelines are given in Noyes (1997) and Bishop (1997). Training the neural network with exact (i.e., noiseless) data is straightforward. The data set is divided into two sets: a training set and a test or validation set, which the neural network will never see in training mode. It is taught with the training set. When its output has reached a satisfactory level of performance with this set, its accuracy is evaluated on the test set.

Training a neural network with noisy data, e.g., data that contain measurement inaccuracies, is a more delicate operation. The network has to learn the main features of the data without learning the noise in it. In neural network terms, one would say that the network should not overtrain. One method of achieving this is to use a network of the "right size", i.e., large enough for it to be able to learn the relevant information, but too small for it to be able to learn the noise. Another technique often used, when only noisy data are available, consists in periodically turning off the neural network learning, and testing its performance with the validation data set. Its error on this set is monitored and the training is ended when this error stops decreasing. In the present study, in which we have both exact and noisy data available, we adopted the following strategy. We used the exact data to look for a neural network that would be as simple as possible, while being able to learn the fluid dynamics function **f**, with the desired precision. We then used a neural network with the same architecture for training on the noisy data. Evaluation of the network's performance was done with both an exact and a noisy test set.

## 5
## The data

Studies such as this, which propose to assess the precision of data analysis methods in PTV and PIV, are best conducted with constructed data instead of experimental data. Indeed, knowing the exact mathematical solution as well as the exact statistical properties of the data allows one to compute exactly the errors made by the data-processing methods. On the other hand, when using real experimental data, the position of the particles as well as their velocities is not known. It then becomes very difficult and would require much more sophisticated statistical analysis to evaluate the efficiency of data processing methods. Because of this, using artificial data is a common procedure, used for example in Luff et al. (1999), Song et al. (1999), Carosone et al. (1995), Guézennec et al. (1994), Westerweel (1994) and Huang et al. (1993). We have considered hereafter three two-dimensional fluid flows, for which the equations of motion are readily available, e.g., in Warsi (1992). We considered the photographs to be squares with sides equal to one, and placed the origin of our coordinate system at the center of this square.

Our first flow is the Poiseuille flow of a viscous fluid between two infinite parallel plates positioned at $y = -0.5$ and $y = 0.5$. The displacements of the fluid particles between time $t$ and $t + \Delta t$ are in the positive $x$-direction, and are given by $D(1 - y^2)$. The maximum displacement $D$, which we took to be 0.075, occurs at the midpoint between the two plates.

Our second flow is that of an inviscid fluid moving in a 90° corner, corresponding to the region where $x \geq -0.5$ and $y \geq -0.5$. It moves from $y = \infty$ toward $x = \infty$ and its velocity vector is proportional to $(x + 0.5, -y - 0.5)$. The fluid particles that are at $(x_1, y_1)$ at time $t_1$ move to $(x_2, y_2)$ at time $t_2$ with $x_2 = (x_1 + 0.5)\, e^{D} - 0.5$ and $y_2 = (y_1 + 0.5)\, e^{-D} - 0.5$ where D is a constant, which we set to be 0.1.

Our third flow is the laminar flow produced by a vortex filament in a viscous fluid at rest, i.e., an Oseen-vortex. The center of the vortex is at the origin of our coordinate system. The streamlines are concentric circles centered on the origin and the angular velocity

$$V_\theta = r\frac{d\theta}{dt} = \frac{Ar}{t^2}\exp\left(-\frac{r^2}{Bt}\right)$$

where A and B are some constants. This equation is readily integrated for $\theta$ and yields

$$\theta(t_2) = \theta(t_1) + \frac{AB}{r^2}\left[\exp\left(-\frac{r^2}{Bt_2}\right) - \exp\left(-\frac{r^2}{Bt_1}\right)\right] .$$

Our data was constructed with $A = 1$ and $B = 1$, $t_1 = 0.1$ and $t_2 = 0.1085$ so that $\theta(t_2) - \theta(t_1) \leq \pi/4$.

Our exact data are constructed as follows. We start with a set of uniformly distributed random points in the unit square $\mathbf{x}_i$ for $i = 1, \ldots, N$, as the geometric centers of the particles of the first photo. We then calculate the position $\mathbf{X}_i$ for $i = 1, \ldots, N$, that these points would have $\Delta t$ later, according to the fluid equation of motion. This yields the set of points for the second photo. We chose the parameters so that the displacements would be as large as possible, while remaining compatible with realistic PTV data. In so doing, we wanted to make the approximation problem difficult, by ensuring that the function **f** would be as far from the identity as possible. Figure 5 illustrates the nature of the data we have used. It shows the displacements undergone by 200 randomly positioned particles in the three flows we have considered. Note that the two constructed point sets did not have to be matched by some algorithm because by construction, the point $\mathbf{X}_i$ in the second set corresponds to the point $\mathbf{x}_i$ with the same index, in the first set.

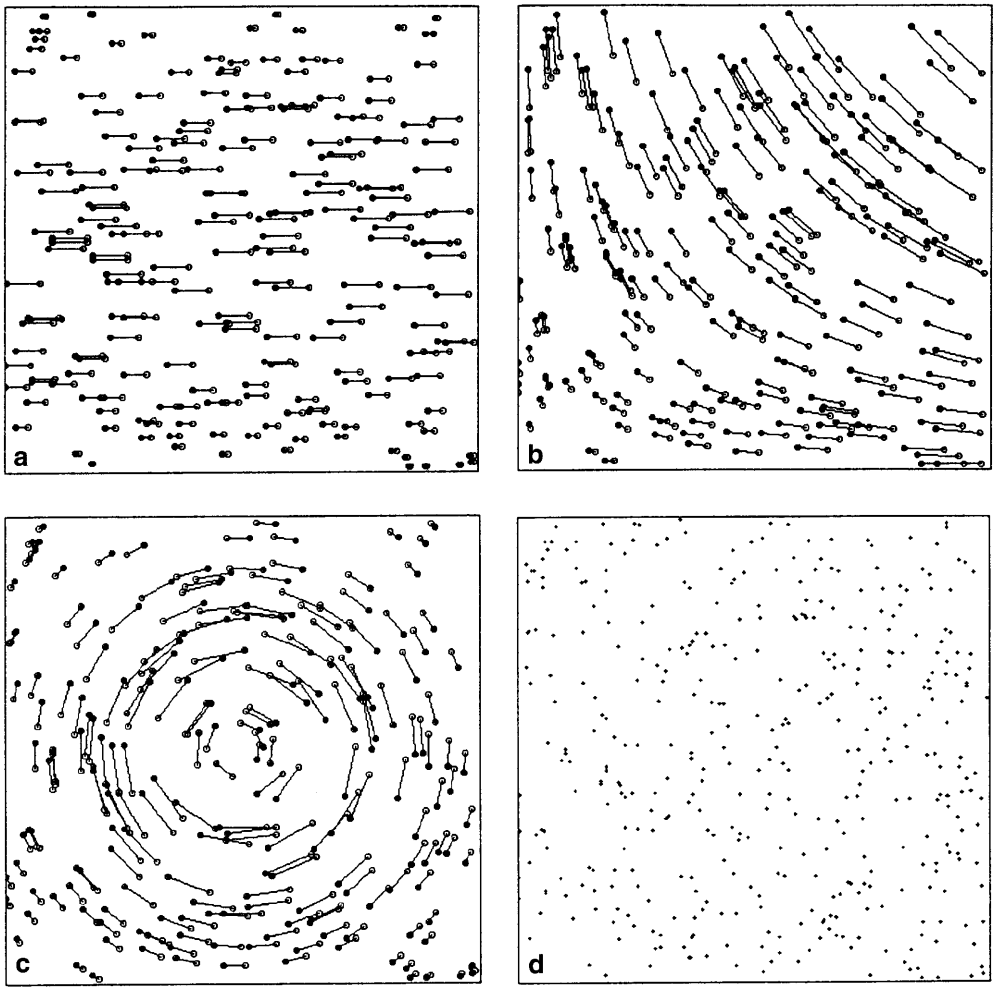


**Fig. 5a–d.** Theoretically calculated displacements of 200 randomly positioned tracer-particles in the three types of flows considered in this article. **a** Poiseuille flow in a channel between two planes; **b** inviscid flow in the 90° corner bounded by the left hand and bottom sides of the picture and **c** flow produced by a vortex filament at the center of the picture. **d** Initial position of the geometric centers of 400 particles in a complete training data set

We then constructed our noisy data sets by adding some small random vectors to the particle position vectors in the exact data sets. Thus, the noisy data were designed to reproduce the data obtained in a PTV experiment, after the outlier displacement vectors, i.e., the obvious mismatches, have been removed. We took the components of these random error vectors to be zero-mean normally distributed numbers, with a standard deviation $\sigma$ of 0.001. If we consider that our photographs are digitized in $512 \times 512$ pixels, the size of one pixel would be about $1.953e-3$ and thus $\sigma \approx 0.5$ pixel. The errors added to the data are then such that there is a probability of 1/3 of there being a one-pixel error in the horizontal coordinate, as well as a probability of 1/3 for the same error in the vertical coordinate in the position vector of each tracer-particle. The size of these errors is well within the normal range of values quoted by other authors, as we recalled in Sect. 1.1.

## 6
## The tests

For each type of fluid flow, the tests were conducted with two sets of data: one with 600 pairs of points of exact data, and one with 600 pairs of points of noisy data. Of each set, 400 pairs were used to train the neural network and the other 200 pairs were kept as testing sets, and were never shown to the neural networks in learning mode. The nature of the sampling of the unit square that 400 points provide can be visualized in Fig. 2d. The average distance between such points is 0.05. For PTV photographs that contain about 100 to 200 tracer-particle images, the data we use for training the neural network would correspond to only 2–4 pairs of PTV photographs.

A series of tests were made with three- and four-layer neural networks having various numbers of neurons in the hidden layer. Although the three-layer networks proved adequate, they were in general appreciably slower in learning the fluid flows than neural networks with four layers. We found that, with the same total number of parameters, i.e., weights and thresholds, the latter networks converged faster. Neural networks with 36 neurons, configured in four layers as 2–16–16–2, generally gave excellent results, and the results we report hereafter are for such networks. Each training session started by initializing the weights to random values. Because of this, the number of cycles required for training a neural network can vary appreciably from one run to the other. The number of cycles we give hereafter only provides an order of magnitude; they are not necessarily very good learning times.

When the input vector is $\mathbf{x}$ and the output vector provided in the data set is $\mathbf{X}$, we define the output error of the neural network $E(\mathbf{x})$ to be the Euclidean distance between $\mathbf{F}(\mathbf{x})$ and $\mathbf{X}$. We denote by $E_{\text{av}}$ and $E_{\max}$ respectively the average and the maximum value of $E(\mathbf{x})$ over all inputs $\mathbf{x}$

in the data set considered. We define a tolerance parameter ε, which we set equal to 1.953e − 3 (about 1/512, which is the size of one pixel). If $\mathbf{F}(\mathbf{x}) = (x_{\text{net}}, y_{\text{net}})$ and $\mathbf{X} = (X, Y)$, we shall consider that the network output is within tolerance of the provided output if both $|x_{\text{net}} - X|$ and $|y_{\text{net}} - Y|$ are smaller than this value. We define the success rate SR of the neural network as the percentage of its outputs that satisfies this condition when all the inputs are presented to it.



### 6.1 Training on exact data

In the first series of tests, the neural network was trained with 400 pairs of points of exact data. After each cycle through this data set, the two errors $E_{\text{av}}$ and $E_{\text{max}}$ and the success rate were calculated. The training was stopped when the success rate reached about 95%. In most cases, a higher success rate could have been achieved on the training set, but we were satisfied with the precision and did not want to risk over-training the network. The results obtained are presented in Table 1. The vortex flow is seen to require a much larger training time than the other two flows. Some of our tests have indicated that a luckier random initialization of the weights could result in shorter training times, such as about 150,000 cycles. Nevertheless, the above-mentioned fact remains true.

After its training on exact data, the performance of the neural network was tested on the 200 pairs of points of both the exact and the noisy data test set. After a cycle through the test set, with the network's learning ability turned off, the two errors $E_{\text{av}}$ and $E_{\text{max}}$ and the success rate were calculated. The results obtained are shown in Table 2. As is obvious, the performance of the neural network on the exact data test set is as good as its performance on the training set. The average errors are of the same order of magnitude and the success rates are essentially identical. This indicates that the neural network can generalize or interpolate well from its training data. Its performance on the noisy data test set gives an indication of the best that should be expected for such a neural network trained on noisy data.

### 6.2 Training on noisy data

In the second series of tests, the neural network was trained with 400 pairs of points of noisy data, i.e., the data that included simulated measurement errors. We used the same procedure as for the exact data. Training was periodically stopped and the performance of the neural network assessed with the exact data test set. It was ended when the network performance reached a satisfactory level. The results obtained are presented in Table 3. One notices that the average errors, when tested on noisy data, are about the same as those made by networks trained on exact data (see Table 2). The success rates are also within about 10% of each other. We interpret this as a good indication that the learning of the neural networks cannot significantly be improved past the point presently reached.

In the same way as when the neural networks were trained with exact data, the networks were then tested on the 200 pairs of points in the exact and in the noisy data test sets. The results obtained are shown in Table 4. We note that their performance is as good as that of neural networks trained on exact data. In fact, their average errors and success rates are somewhat better, which would be surprising were it not for the studies mentioned above in Sect. 3.1 that indicate that adding noise to a neural network training data results in improved performance.

As a convincing demonstration of the excellent quality of the approximation realized by the neural networks, we have used them to plot whole streamlines for the flow in a 90° corner and for the vortex flow. For the Poiseuille flow, the streamlines are simply straight lines. We thus had the neural network perform the more difficult task of plotting

**Table 1.** The *second column* shows the number of cycles for which the neural network was trained. $E_{\text{av}}$ and $E_{\text{max}}$ represent the average and maximum Euclidean distance between the network output vector and the output value provided in the training data set. The *last column* shows the success rate, i.e., the percentage of network outputs that are within the specified tolerance of the outputs provided in the data set

| N.N. training on exact data | | | | |
|---|---|---|---|---|
| Flow | Number of cycles | $E_{\text{av}}$ | $E_{\text{max}}$ | SR (%) |
| Channel | 6,500 | 0.000963 | 0.005549 | 97.75 |
| Corner | 10,000 | 0.001045 | 0.003461 | 97.25 |
| Vortex | 500,000 | 0.001093 | 0.003982 | 95.5 |

**Table 2.** Performance of the neural networks trained with exact data, evaluated with the exact and the noisy data test sets. $E_{\text{av}}$ and $E_{\text{max}}$ are respectively the average and maximum Euclidean distances between the vector output by the network and the vector output provided in the test set. The last column shows the success rate, i.e., the percentage of points for which the neural network output is within the specified tolerance of the output provided in the test set

| Evaluation of N.N. trained on exact data | | | | | | |
|---|---|---|---|---|---|---|
| | Exact data | | | Noisy data | | |
| Flow | $E_{\text{av}}$ | $E_{\text{max}}$ | SR (%) | $E_{\text{av}}$ | $E_{\text{max}}$ | SR (%) |
| Channel | 0.001762 | 0.016889 | 96.0 | 0.002658 | 0.017839 | 61.5 |
| Corner | 0.001165 | 0.005623 | 96.0 | 0.003106 | 0.005418 | 58.0 |
| Vortex | 0.001555 | 0.013096 | 94.0 | 0.002724 | 0.013429 | 55.0 |

**Table 3.** The first column shows the number of cycles for which the neural network was trained with noisy data. $E_{\text{av}}$ and $E_{\text{max}}$ represent the average and maximum distance between the network output vector and the output value provided in the training data set. The last column shows the success rate, i.e., the percentage of points for which the neural network output is within the specified tolerance of the output provided in the data set

| N.N training on noisy data | | | | |
|---|---|---|---|---|
| Flow | Num. cycles | $E_{\text{av}}$ | $E_{\text{max}}$ | SR (%) |
| Channel | 6,500 | 0.002107 | 0.005725 | 67.5 |
| Corner | 6,000 | 0.002091 | 0.005465 | 64.4 |
| Vortex | 652,500 | 0.002073 | 0.005055 | 67.5 |

**Table 4.** Performance of the neural networks trained with noisy data on the exact and the noisy data test sets. $E_{av}$ and $E_{max}$ are respectively the average and maximum Euclidean distances between the vector output by the network and the vector provided in the test set. The last column shows the success rate SR, i.e., the percentage of points for which the neural network output is within the specified tolerance of the output provided in the test set

| Evaluation of N.N. trained on noisy data | | | | | | |
|---|---|---|---|---|---|---|
| | Exact data | | | Noisy data | | |
| Flow | $E_{av}$ | $E_{max}$ | SR (%) | $E_{av}$ | $E_{max}$ | SR (%) |
| Channel | 0.001078 | 0.009267 | 97.5 | 0.002332 | 0.010116 | 64.0 |
| Corner | 0.000659 | 0.002682 | 98.5 | 0.002054 | 0.005404 | 70.0 |
| Vortex | 0.001384 | 0.007471 | 92.0 | 0.002548 | 0.006658 | 51.5 |

the flow profile, which shows the deformation of a plane of fluid that starts at the LHS of the picture. The neural networks were provided with the starting point on the boundary of the unit square, and then had to produce the whole sequence of points making the lines by iteration. This is a good test of the accuracy of the approximation obtained, because, in this process, the errors are accumulated from point to point. Figure 6 shows the theoretical lines, as continuous lines and the lines produced by the neural networks as dashed lines. The neural networks used were those trained on the data with simulated experimental errors. The lines produced by the neural networks trained with exact data are too similar to be worth including.

## 7 Conclusion

Once particle images have been paired and the obvious outliers have been removed, there is further need for data processing in PTV. The data are still left with some random noise, holes where the matching was unsuccessful and, of course, the empty regions between the measured displacements – hence, the necessity for data smoothing and interpolating algorithms. Most of the time, smoothing is performed with filters corresponding to the convolution of the data with a Gaussian or a finite impulse function. Interpolation is essentially always done locally, for small patches, using splines or linear regression.

We have described simple neural networks that can perform both of these standard functions. As is evident in our discussion, the neural networks do exactly the same set of mathematical operations as the classical algorithms. They provide however a very important advantage in that they perform these operations in parallel. We have shown that for the processing of PTV data, this results in speed-up factors in the tens of thousands.

We have then described the type of non-linear regression that neural networks can also perform, which is much more powerful than linear regression. We have cited a general theorem that guarantees that a neural network exists that can approximate any piecewise continuous function to any specified precision. However, this is only an existence theorem. It gives no indications as to how easy or difficult it would be to come up with such a network in a particular field of application. In the present study, we have demonstrated that it is actually very easy to produce neural networks that realize the fluid dynamics function that maps the fluid particle position at time $t$ to that at time $t + \Delta t$.

To train our neural network, we constructed artificial data for three different non-trivial ideal fluid flows for which the exact mathematical description is known. One is a Poiseuille flow in a channel, one is a flow in a 90° corner, and the other one a flow in an Oseen-vortex. These data consist of coordinate pairs $(\mathbf{x}_i, \mathbf{X}_i)$ that correspond to the correctly matched positions of the $i$th particle in two successive PTV pictures. We constructed some data sets in which the image positions were without inaccuracies and some in which we have added small random errors to



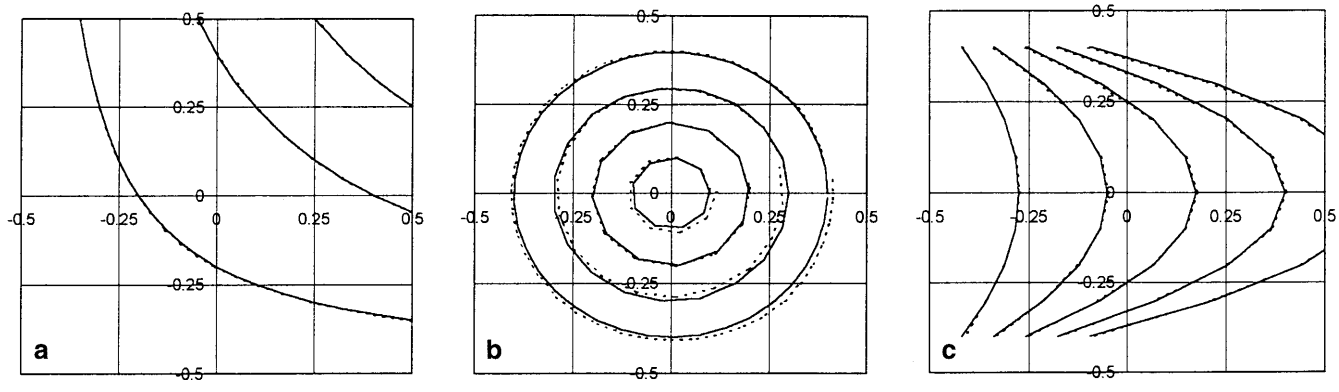


**Fig. 6. a** Three streamlines for the flow in a 90° corner. The *dashed lines* are produced by the neural network by iteration starting from a point given at the top of the picture. The *solid lines* are the corresponding theoretically calculated streamlines. **b** Four streamlines for the vortex flow, that start and return to points on the $x$-axis after circling the origin. The *dashed lines* are those that the neural network produced by iteration. The *solid lines* are the theoretically calculated lines. **c** Flow profile for the Poiseuille flow in a channel. The lines represent the successive surfaces into which a plane of the fluid, which starts at the LHS of the picture, is deformed. The surfaces are obtained by plotting the positions of points that start at $x = -0.5$ after each three successive displacement steps. The *dashed lines* are those produced by the neural network and the *solid lines* are calculated from the equation of motion

them in order to simulate experimental data obtained after the removal of outliers. For our neural networks, we used a commonly available neural network simulation software. We selected "quickprop" among the standard training algorithms offered as options and kept all the default parameters proposed by the software. As network structure, we started with simple three-layer networks. These already yielded reasonable results. We then tried four-layer networks and found, after two or three attempts, that a network with its neurons distributed in the layers as 2–16–16–2, produced excellent results. We have thus clearly demonstrated that it is easy to find simple back-propagation neural networks that can learn to represent non-trivial fluid flows.



We have also demonstrated that the neural networks that are trained on data with measurement errors can filter these out and form a representation of the fluid flow that is as precise as when they are trained on exact theoretical data. The results we obtained by training with exact data are shown in Table 2 and those with noisy data in Table 4. One can see that, when tested with data corresponding to the exact analytical solution, the average errors and success rates of networks trained with noisy data are even better than those trained on exact data. This corroborates the theory mentioned in Sect. 3.1 to the effect that adding noise to a neural network training data improves its performance. This property makes neural networks an ideal tool for the post-processing of experimental PIV data.

As can be seen in Tables 1 and 3, training the neural networks took comparable numbers of cycles, whether they were trained on exact or noisy data. The differences we found between the two are well within the normal range of variation expected, even for the same network, when a gradient descent algorithm is used. This variation can simply be due to the fact that the network weights are assigned random values initially.

The outputs of our neural networks are within less than 0.002 of the correct outputs, for at least 95% of the inputs. This is 25 times more precise than the average distance between the data samples they have seen in training, which is 0.05. These results corroborate the theory in indicating that indeed arbitrarily high precision is achievable.

In our tests, we chose to consider data sets with as few as 400 data points, which correspond to about 2–4 pairs of PTV photographs. This allowed us to demonstrate the interpolating power of the neural network. When using a correlation method in PIV, this would correspond to a particularly sparse array of interrogation windows of dimension $20 \times 20$. Data at points in arrays of $50 \times 50$, and much more, are however easily obtained. In this case, the point spacing would be $4 \times 10^{-4}$ and less, as compared to $5 \times 10^{-2}$ in the present study (for photographs of dimensions normalized to one). The quality of the neural network interpolations is expected to improve by a corresponding factor in such situations.

We have demonstrated more concretely the good quality of the fluid flow representations our neural networks achieved by showing that they can draw complete streamlines or flow profiles, with high precision. This ability makes them particularly well suited for fluid flow visualization.

Finally, we note that it is not very significant to compare the times required for training software-simulated neural networks to the processing times required by algorithms designed for classical sequential computers. The pertinence of this remark becomes obvious when one considers the neural networks, proposed in Sect. 2.1 and 2.2, to implement standard convolution filters and linear regression. Since these perform exactly the same set of mathematical operations as the standard sequential algorithms, simulating them in software on a sequential computer will never exhibit any of their advantages. They will take the same time, if not longer. The processing time speed-up factors in the tens of thousands, which they can provide, will only be revealed when they are realized on parallel-computing hardware. Studies such as the present one are exploratory and serve to determine neural networks that can perform a particular task in a particular domain. Once they are completed, they would normally be followed by the implementation of the neural networks on special parallel hardware, PC boards or VLSI (see, e.g., Section A 2.2.3 of Werbos 1997), where they show their full data-processing power. Note nevertheless that, as far as performing non-linear regression is concerned, even when considered as sequential numerical techniques, neural network algorithms are faster than the methods generally used in conventional statistics (see Section A 2.3.3.1 of Werbos 1997).

Further work will involve trying to find neural network architectures and parameters that will optimize training times. The efficiency of other types of networks in representing fluid dynamics functions, such as radial basis function networks, should also be compared to that of feed-forward networks. Studies should be made to test the abilities of neural networks with experimental PTV data. The same advantages that neural networks provide in the post-processing of PTV data will also be seen when they process PIV data obtained by correlation methods.

## References

**Adrian RJ** (1991) Particle-imaging techniques for experimental fluid mechanics. Ann Rev Fluid Mech 23: 261–304

**Almeida LB** (1997) Multilayer perceptrons. In: Fiesler E, Beale R (eds) Handbook of neural computation. Institute of Physics/ Oxford University Press, Oxford, pp C1.2: 1–C1.2: 30

**Astola J; Haavisto P; Neuvo Y** (1990) Vector median filters. Proc IEEE 78: 678–689

**Battiti R** (1992) First- and second-order methods for learning: between steepest descent and Newton's method. Neural Comput 4: 141–166

**Bishop CM** (1997) Neural networks: a pattern recognition perspective. In: Fiesler E, Beale R (eds) Handbook of neural computation. Institute of Physics/Oxford University Press, Oxford, pp B6.1: 1–B6.5: 2

**Carosone F; Cenedese A; Querzoli G** (1995) Recognition of partially overlapped particle images using the Kohonen neural network. Exp Fluids 19: 225–232

**Cybenko G** (1989) Approximation by superpositions of a sigmoidal function. Math Control Signals Systems 2: 303–314

**Fahlman SE** (1989) Fast training variations on back-propagation: an empirical study. In: Touretzky D, Hinton G, Sejnowski T (eds) Proceedings of the 1988 Connectionist Models Summer School, Pittsburgh. Morgan Kaufmann, San Mateo, Calif, pp 38–51

**Funahashi K-I** (1989) On the approximate realization of continuous mappings by neural networks. Neural Networks 2: 183–192

**Guézennec YG; Brodkey RX; Trigui N; Kent JC** (1994) Algorithms for fully automated three-dimensional particle tracking velocimetry. Exp Fluids 17: 209–219
**Guézennec YG; Kiritsis N** (1990) Statistical investigation of errors in particle image velocimetry. Exp Fluids 10: 138–146
**Hardy RL** (1971) Multiquadric equations of topography and other irregular surfaces. J Geophys Res 76: 1905–1915
**Hartman J; Köhler J; Stolz W; Flögel HH** (1996) Evaluation of instationary flow fields using cross-correlation in image sequences. Exp Fluids 20: 210–2217
**Hassoun MH** (1995) Fundamentals of artificial neural networks. MIT Press, Cambridge, Mass
**Hebb D** (1949) The organization of behavior. Wiley, New York
**Hecht-Nielsen R** (1989) Neurocomputing. Addison-Wesley, Reading, Mass
**Hornik K; Stinchcombe M; White H** (1989) Multilayer feedforward networks are universal approximators. Neural Networks 2: 359–366
**Huang HT; Fiedler HE; Wang JJ** (1993) Limitation and improvement of PIV. Exp Fluids 15: 263–273
**Labonté G** (1999) A new neural network for particle-tracking velocimetry. Exp Fluids 26: 340–346
**Landreth CC; Adrian RJ** (1990) Measurement and refinement of velocity data using high image density analysis in particle image velocimetry. In: Adrian RJ et al (eds) Applications of laser anemometry to fluid mechanics. Springer, Berlin Heidelberg New York, pp 484–497
**Lourenco L; Krothapalli A** (1995) On the accuracy of velocity measurements with PIV. Exp Fluids 18: 421–428
**Luff JD; Drouillard T; Rompage AM; Linne MA; Hertzberg Jr** (1999) Experimental uncertainties associated with particle image velocimetry (PIV) based vorticity algorithms. Exp Fluids 26: 36–54
**McCulloch WS; Pitts W** (1943) A logical calculus of the ideas immanent in nervous activity. Bull Math Biophys 5: 115–133
**Noyes JL** (1997) Neural network training. In: Fiesler E, Beale R (eds) Handbook of neural computation. Institute of Physics/ Oxford University Press, Oxford, pp B3.5: 1–B3.5: 6
**Poggio T; Girosi F** (1990) Networks for approximation and learning. Proc IEEE 78: 1481–1497
**Schwartz DB; Samalam VK; Solla SA; Denker JS** (1990) Exhaustive learning. Neural Comput 2: 374–385
**Song X; Yamamoto F; Iguchi M; Murai Y** (1999) A new tracking algorithm of PIV and removal of spurious vectors using Delaunay tessellation. Exp Fluids 26: 371–380
**Sun JH; Yates DA; Winterbone DE** (1996) Measurement of the flow field in a diesel engine combustion chamber after combustion by cross-correlation of high-speed photographs. Exp Fluids 20: 335–345
**Tamura S; Waibel A** (1988) Noise reduction using connectionist models. Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing. IEEE Press, New York, pp 553–556
**Troudet T; Merrill WC** (1990) Neuromorphic learning of continuous-valued mappings from noise-corrupted data. National Aeronautics and Space Administration Technical Memorandum 4176, USA
**Warsi ZUA** (1992) Fluid dynamics, theoretical and computational approaches. CRC Press, Boca Raton, Fla
**Werbos PJ** (1974) Beyond regression: new tools for prediction and analysis in the behavioral sciences. PhD dissertation, Applied Math., Harvard University
**Werbos PJ** (1997) Why neural networks? In: Fiesler E, Beale R (eds) Handbook of neural computation. Institute of Physics/ Oxford University Press, Oxford, pp A2.1: 1–A2.3: 6
**Westerweel J** (1994) Efficient detection of spurious vectors in particle image velocimetry data. Exp Fluids 16: 236–247
**Widrow B; Hoff ME Jr** (1960) Adaptive switching circuits. IRE Western Electric Show and Convention Record, Part 4, pp 96–104
**Widrow B; Stearns SD** (1985) Adaptive signal processing. Prentice Hall, Englewood Cliffs, NJ
**Willert CE; Gharib M** (1991) Digital particle image velocimetry. Exp Fluids 10: 181–193